Anton Konstantinos Pinnis

METEOR-POHJAISEN WEB-SOVELLUKSEN KEHITTÄMINEN

METEOR-POHJAISEN WEB-SOVELLUKSEN KEHITTÄMINEN

Anton Konstantinos Pinnis
Opinnäytetyö
Kevät 2016
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehityksen sv

Tekijä(t): Anton Konstantinos Pinnis
Opinnäytetyön nimi: Meteor-pohjaisen web-sovelluksen kehittäminen
Työn ohjaaja(t):Kari Laitinen
Työn valmistumislukukausi ja -vuosi: Kevät 2016 Sivumäärä: 48

Työn tavoitteena oli tutustua Meteor-sovelluskehykseen ja kehittää sillä web-sovelluksen prototyyppi.

Työ aloitettiin käymällä läpi Meteorin tutustumismateriaalia, jolla saatiin ensivaikutelma Meteorin periaatteista, kuten web-sovelluksen kehitys yhdellä ohjelmointikielellä, toimintojen lisääminen hyödyntämällä Meteorin paketteja ja NoSQL-tietokannan käyttö relaatiotietokannan sijaan. Web-sovelluksen kehittäminen aloitettiin käyttämällä ensimmäisessä vaiheessa opittuja periaatteita.

Kehitetyn web-sovelluksen tarkoitus on toimia oppimisympäristönä, jolla voi harjoitella pörssiosakkeiden sijoittamista. Työssä saatiin web-sovellukseen toteutettua esimerkiksi käyttäjätilijärjestelmä, monikielisyys ja navigaatio sivujen välillä.

Asiasanat: Meteor, Web-sovellus, Web-ohjelmointi, Node.js, NoSQL

ABSTRACT

Oulu University of Applied Sciences
Information Technology and Telecommunications, Software Development

---

Author(s): Anton Konstantinos Pinnis
Title of thesis: Web development with Meteor
Supervisor(s): Kari Laitinen
Term and year when the thesis was submitted: Spring 2016 Pages: 48

The topic of this thesis was to familiarize with Meteor and use it to develop a prototype web application.

The work started by going through all Meteor related learning material to learn Meteor's principles like web application programming only with one language, adding features to web application by using Meteor's packages and use of NoSQL database instead of Relational database. The work continued with the development of the web application by using Meteor's principles.

The developed prototype application will be used as a learning environment in which it is possible to practice investing in stocks. The work ended with a web application, which included important features, such as User Account System, multi-language support and navigation between web pages.

Keywords: Web application, Web programming, Meteor, Node.js, NoSQL

SISÄLLYS

SANASTO

Blaze	Meteorin sisään rakennettu kirjasto käyttöliittymän reaktiivista päivitystä varten
CSS	Cascading Style Sheets, ulkoasu HTML-kielellä kuvatulle rakenteelle
DDP	Distributed Data Protocol, protokolla datan välittämiseen
Helper	Funktio, jolla välitetään tietoa käyttöliittymälle
JSON	JavaScript Object Notation, avoimen standardin tiedostomuoto
HTML	Hyper Text Markup Language, hypertekstin merkintäkieli
JavaScipt	Web-ympäristössä käytettävä ohjelmointikieli
Meteor	Avoimen lähdekoodin sovelluskehys
MongoDB	Avoimen lähdekoodin tietokanta
MySQL	Relaatiotietokantaohjelmisto
Node.js	JavaScriptiin perustava kehitysalusta

1 JOHDANTO

Opinnäytetyön aiheena oli tutustua Meteor-sovelluskehykseen ja kehittää sillä web-sovelluksen prototyyppi. Työn tilaajana oli Suomen harjoitusyritysten keskus – FINPEC. FINPEC on osa Oulun seudun ammattiopistoa ja tarjoaa oppilaitoksille yrittäjyyskasvatuksen palveluja. Harjoitusyritystoiminnan viitekehys sisältää mm. web-sovelluksen.

Työ aloitettiin käymällä ensin läpi kaikki tutustumismateriaali Meteorista ja tekemällä harjoituksia, jotta opittiin Meteorin web-sovelluksen kehittämisen periaatteita, kuten JavaScriptin käyttö asiakaspuolella ja palvelinpuolella, käyttöliittymän päivitys DDP-protokollalla, reaktiivisuus datan muuttuessa ja MongoDB-tietokannan käyttö perinteisen relaatiotietokannan sijaan.

Työtä jatkettiin aloittamalla prototyypin kehittäminen. Tavoitteena oli saada mahdollisimman paljon toimintoja lisättyä web-sovellukseen käyttämällä Meteorin tarjoamia tekniikoita, projektin aikataulua noudattaen.

Kehitetyn web-sovelluksen tarkoitus on opettaa opiskelijoille, mitä pörssin sijoittaminen käytännössä tarkoittaa. Sovelluksella voi kokeilla sijoituspäätöksia virtuaaliympäristössä.

2 WEB-TEKNIIKKA

2.1 World Wide Web

World Wide Web (WWW) on internet-palvelimien järjestelmä, joka tukee varsinkin dokumenttien esittämistä. Dokumentit on kuvattu hypertekstikielellä nimeltään HTML (HyperText Markup Language), joka tukee yhteyksiä erilaisiin dokumentteihin. Tämä antaa mahdollisuuden siirtyä yhdestä dokumentista toiseen napsauttamalla merkittyjä alueita. WWW:n järjestelmään päästään erilaisilla sovelluksilla kuten Firefox tai Microsoftin Internet Explorer. (1.)

WWW koostui 1990-luvun alussa verkkosivustoista, jotka olivat pelkästään tietoarkistoja, jotka sisälsivät staattisia dokumentteja. Selaimet keksittiin näiden tietojen hakemiseen ja näyttämiseen käyttäjälle. Tiedon siirtäminen oli yksisuuntainen palvelimelta selaimeen, ja nettisivustoissa jokaista käyttäjää kohdeltiin samalla tavalla, koska ei ollut kirjautumista nettisivun tietokantaan. Turvallisuusuhat aiheutuivat pääosin verkkosivuston ohjelmiston tietoturva-aukoista. Jos hyökkääjä saisi pääsyn nettisivun tiedostoihin, hän ei saisi sen enempää henkilökohtaista informaatiota kuin tavallisella käyttäjällä. (2, s. 71–72.)

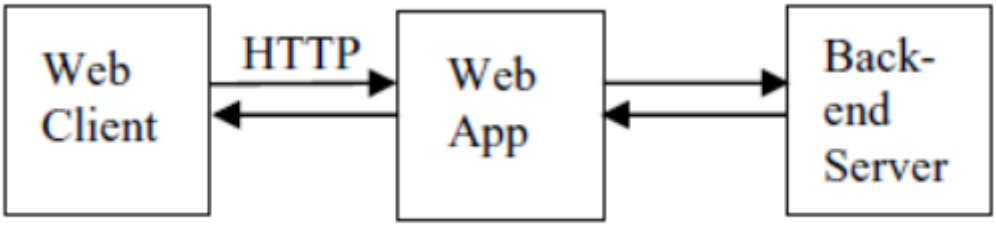
2.2 Web-sovellus

Web-sovellus on mikä tahansa sovellus, joka käyttää selainta asiakasohjelmana. Sovellus voi olla niin yksikertainen kuin viestitaulu, foorumi nettisivulla tai monimutkaisempi tekstinkäsittelyohjelma. (3.)

Asiakasohjelma-termiä käytetään asiakas-palvelinjärjestelmässä kuvaamaan ohjelmaa, jota käyttäjä hyödyntää käyttäessään sovellusta. Asiakas-palvelinjärjestelmässä käyttäjät jakavat tietoja toisille käyttäjille tai lisäävät tietoja tietokantaan. Yleensä asiakasohjelmaa käytetään tiedonsyöttäjänä ja palvelinta tietojen tallentamiseen. (3.)

Web-sovelluksilla kevennetään kehittäjän työtä vähentämällä tarvetta kehittää samaa asiakasohjelmaa useammalle alustalle. Asiakasohjelman ajaminen selaimessa antaa mahdollisuuden ajaa sen useamassa käyttöjärjestelmässä. (3.)

Web-sovellukset käyttävät kehityksessä yleensä palvelinpuolelta ohjelmointikieliä kuten ASP tai PHP ja asiakaspuolelta yhdistelmää JavaScript, HTML ja CSS. Asiakaspuoli hoitaa tietojen näyttämistä käyttäjälle ja palvelinpuoli hoitaa kaikki tallennukset ja tietojen kyselyt tietokannasta (kuva 1). (3.)



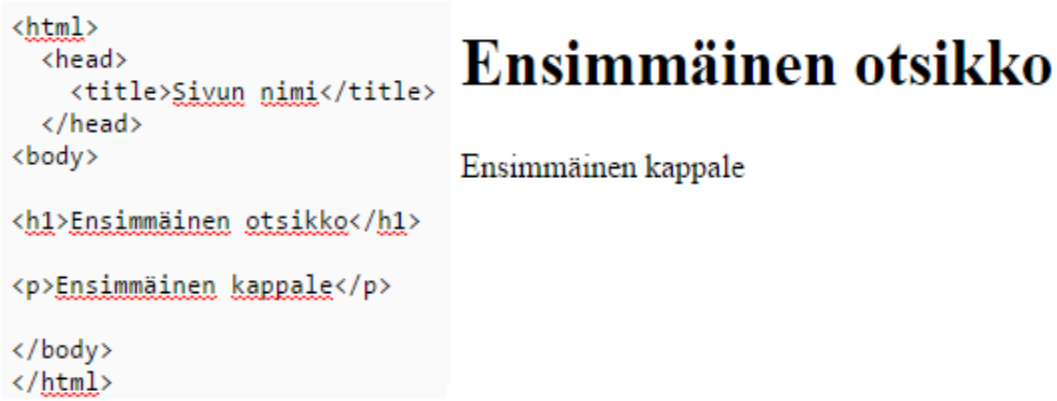
KUVA 1. Web-sovelluksen ympäristö (2, s. 72)

2.3 Web-sivun perusrakenne

2.3.1 HTML-kieli

HTML:ää (HyperText Markup Language) käytetään, kun halutaan luoda digitaalisia dokumentteja, jotka näkyvät WWW:ssä. Jokainen sivu sisältää yhteyksiä eri sivuihin käyttämällä hyperlinkkejä. HTML:n uusin versio tällä hetkellä on HTML5, joka lisäsi kieleen paljon uusia dynaamisia ominaisuuksia. (4.)

HTML varmistaa oikean syntaksin kuville ja tekstille, jotta selain näyttää ne oikeassa järjestyksessä ja muodossa. Ilman HTML:ää selain ei tietäisi esittää tekstejä elementteinä tai ladata kuvia tai muita elementtejä. HTML toimittaa myös sivun perusrakenteen, johon tulee päälle CSS. HTML:n perusrakenne näkyy kuvassa 2. (4.)



KUVA 2. HTML:n perusrakenne

HTML-tiedostot sisältävät monenlaisia elementtejä, joita erotetaan tageilla eli tunnisteilla. HTML:n koodin kirjoittaminen aloitetaan elementillä ”<html>…</html>”.Tagien loppuosassa on aina “/”, joka merkitsee tagin loppua. Head-elementin sisälle voidaan lisätä yleistä tietoa sivusta, kuten sivun nimi, tai linkittää toisia tiedostoja tähän sivuun. Body-elementin sisälle tulee sivun varsinainen sisältö. Otsikoita sivuun lisätään h-elementeillä “<h1></h1>...<h6></h6>”. Numerot merkitsevät otsikoiden kokoa: mitä pienempää numeroa käytetään, sitä isompi teksti tulee näkyviin. Kappaleita sivuun lisätään p-elementillä ” … ”, jolla on pelkästään yksi koko. (4.)

2.3.2 CSS-tyyliohjekieli

CSS (Cascading Style Sheets) hoitaa nettisivun ulkoasun. CSS lisätään HTML:n tiedostoon, jossa on määritetty käytetyt elementit. CSS:n avulla saadaan verkkosivun ulkoasumuotoilut yhteen tiedostoon, jolloin ulkoasun muuttaminen ja yhtenäisenä pitäminen on olennaisesti helpompaa kuin HTML:n keinoilla. Muotoilu voi tapahtua monella eri tavalla, kuten käyttäen erilistä tyyliohjetiedostoa tai laittamalla tyyliohjeet HTML-tiedoston sisään tai elementtien tagien sisälle. Nykyään suositellaan, että HTML:n avulla esitetään vain sivun perusrakenne ja muotoilut tehdään mahdollisimman pitkälle tyylitiedostojen avulla. (5.)

3 JAVASCRIPT-OHJELMOINTIKIELI

JavaScript on järjestelmäriippumaton ja olioperustainen kieli. Se on pieni ja kevyt kieli. HTML:n elementteihin voidaan tehdä muokkauksia JavaScriptillä. (6.)

JavaScript tarjoaa perusobjekteja, kuten Array, Date, Math, ja kielen ytimen osia, kuten operaattoreita ja ohjausrakenteita. JavaScriptin ytimen voi laajentaa moneen eri tarkoitukseen täydentämällä sitä objekteilla. (6.)

Asiakaspuolella JavaScriptia käytetään pääsääntöisesti selaimen ja sen DOMin (Document Object Model) ohjausta varten. Esimerkiksi JavaScriptillä pystyy asiakaspuolella sijoittamaan elementtejä yhdessä HTML:n lomakkeessa ja vastaamaan käyttäjästä aiheutuviin tapahtumiin, kuten hiiren napautuksiin, lomakkeen jättämiseen ja sivun navigaatioon. (6.)

Palvelinpuolella on olemassa JavaScript-pohjaisia kirjastoita ja tekniikoita, joilla voidaan esimerkiksi antaa lupa web-sovellukselle kommunikoida tietokannan kanssa, tarjota jatkuvaa tietoa yhdestä hyödyntämistä toiseen applikaatiossa tai toteuttaa tiedostojen manipulointia palvelimessa. (6.)

3.1 Historia

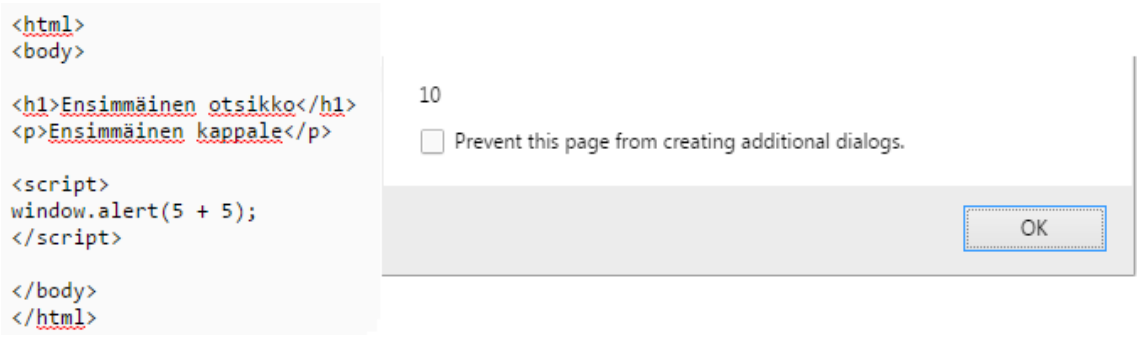
JavaScriptin loi Breindan Eich vuonna 1995 samaan aikaan, kun hän oli töissä Netscape Communicationsilla. JavaScript julkaistiin ensimmäistä kertaa Netscape 2:n kanssa vuoden 1996 alussa. Se aiottiin alun perin nimetä LiveScriptiksi. Se sisältää elementtejä ohjelmointikielistä Java, Scheme ja Self. (7.)

3.2 Perusrakenne

JavaScriptin toteutus on melko samanlainen verrattuna CSS:ään. Koodia voi lisätä sovellukseen kahdella eri tavalla:

1. Ulkopuolinen tiedosto, joka päättyy js:ään
2. Käyttämällä script-tagia HTML:ssä.

Esimerkki toisesta tavasta on kuvassa 3.

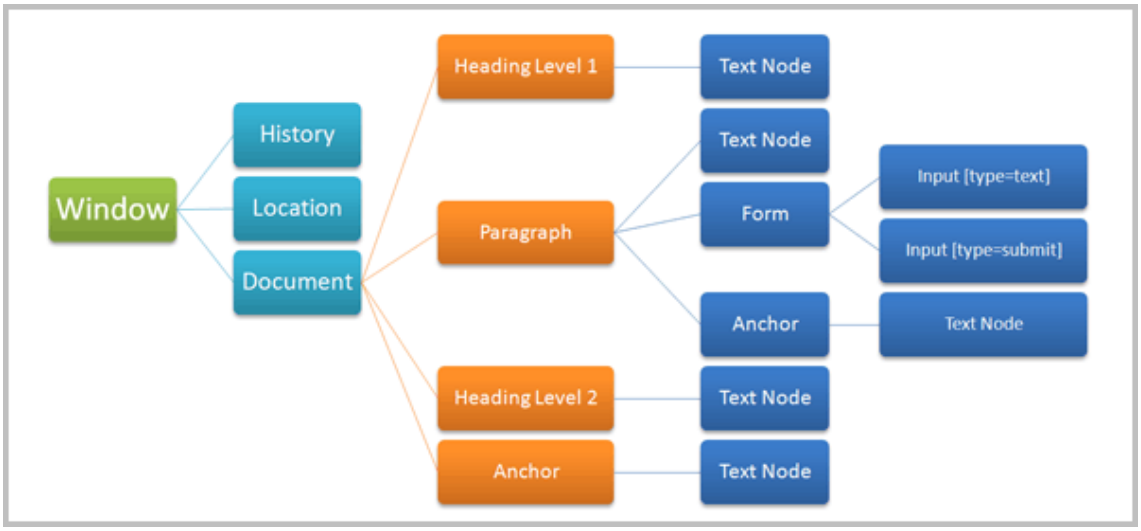


KUVA 3. JavaScriptia HTML:n sisällä

JavaScriptiä on lisätty tagien ”<script></script>” sisälle. Esimerkissä käytettiin funktiota window.alert(), joka luo ruudulle ikkunan ja siihen tässä tapauksessa tekstin 10.

3.3 Document Object Model

DOM tai Document Object Model on ohjelmointirajapinta HTML-dokumentille. Dokumentti sisältää monenlaisia elementtejä (kuva 4), joita voidaan ajatella objekteina. Näillä objekteilla on ominaisuuksia, kuten väri, pituus jne. JavaScriptillä pystytään lukemaan ja muokkaamaan näitä arvoja tai reagoimaan tapahtumiin. (8.)



KUVA 4. DOM-mallin esimerkki (8.)

3.4 Ominaisuuksia

JavaScriptin muuttujat tallentavat dataa kahdella tavalla: kopioimalla ja viittauksella. Kaikki primitiiviset arvot kopioituvat muuttujaan. Primitiivejä ovat merkkijonot, numerot, totuusarvomuuttujat, tyhjät ja määrittelemättömät. Primitiivien tärkein ominaisuus on, että ne tulee osoitettua, kopioitua ja luovutettua funktioihin ja niistä takaisin arvolla. Muut muuttujat, jotka eivät kuulu primitiiveihin, tallentavat viitteen eli osoitteen yhteen objektiin. Viite on osoite objektin sijainnista muistissa. Tämä antaa mahdollisuuden siihen, että kahdella tai useimmalla muuttujalla on käytössä sama objekti käyttämällä pelkästään objektin osoitetta. (9, s. 7.)

Objekteilla JavaScriptissä on kaksi piirrettä: ominaisuudet ja metodit. Näitä yleensä kutsutaan objektin jäseneksi. Ominaisuudet voivat olla primitiivejä tai objektit itse. Metodit ovat funktioita, jotka käyttäytyvät datan mukaan. (9, s. 8.)

Kontekstilla koodissa tarkoitetaan näkyvyysaluetta, jossa koodi suoritetaan. Konteksti voi olla tehokas työkalu ja se on olennainen olio-ohjelmoinnissa. JavaScriptissä pääsy kontekstiin onnistuu komennolla ”this”, joka viittaa suoritettavan koodin kontekstiin. (9, s. 11.)

3.5 jQuery

jQuery on nopea ja suppea JavaScript-kirjasto, jonka loi John Resig vuonna 2006. jQuery on tällä hetkellä suosituin JavaScript-kirjasto, mikä johtuu sen helposta käytöstä ja tehosta lisätä toimintoja nettisivuun. jQuery tarjoaa UI-komponentteja, joilla saadaan nopeutettua käyttöliittymän suunnittelua ja toteutusta ilman aiempaa kokemusta web-sovelluksen kehittämisestä. DOMin manipulointi onnistuu helpommin vähemmällä koodilla ja toimii siinä tapauksessa kun, JavaScript on suljettu selaimessa. (10.)

jQuerylla voi toteuttaa animaatiota nettisivulle käyttämällä kirjaston sisäisiä funktioita yhdessä AJAX-tekniikan kanssa, joka tarjoa nettisivun dynaamisen

päivityksen. Kirjasto toimi kaikissa uusissa selaimissa: IE 6+, Firefox 2.0+, Safari 3+, Opera 10.6+ ja Chrome 8+. (10.)

3.6 Bootstrap

Bootstrap on ohjelmistokehys, joka helpottaa ja nopeuttaa web-sovelluksen kehittämistä. Bootstrapilla kehittäjä keskittyy kunnolliseen HTML-koodin kirjoittamiseen ilman suurta kokemusta CSS:stä tai JavaScriptistä. (11.)

Boostrapilla on monta tärkeää ominaisuutta, kuten esimerkiksi seuraavat:

1. Bootstrap nopeuttaa elementtien lisäämistä ja muokkaamista nettisivulle valmiilla CSS- ja JavaScript-paketeilla.
2. Saadaan aikaan dynaaminen sivunrakenne, joka sijoittaa uudestaan elementit näytön koon muuttuessa.
3. Projekti pysyy johdonmukaisena.
4. Kotisivulla voi valita tarvittavat tyylit omalle nettisivulle ilman ylimääräisiä ominaisuuksia, joita ei tarvita.
5. Boostrapia päivitetään usein ja sillä on paljon seuraajia. (11.)

Frontend-ohjelmistokehyksellä säästetään aikaa käyttöliittymän kehityksestä toimintojen kehittämiseen.

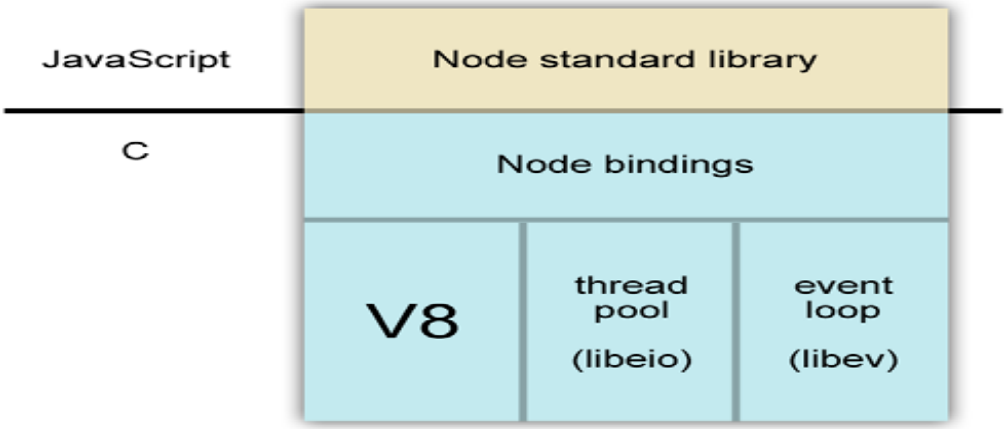
4 NODE.JS-KEHITYSALUSTA

Node.js perustuu Googlen V8-JavaScript-moottoriin. Komentoja kirjoitetaan JavaScript-ohjelmointikielellä, ja V8-moottori kääntää sen konekieleksi suoritusta varten. (12, s. 2.)

Node.js antaa mahdollisuuden kirjoittaa kokonaan palvelinpuolen koodin JavaScriptillä. Sillä saadaan tehtyä web-palvelin, palvelinpuolen skriptit ja jokainen tuettava web-sovelluksen toiminnallisuus. (12, s. 2–3.)

Node.js koostuu pääosiin kolmesta osasta (kuva 5):

- V8 on Googlen JavaScript-moottori, jota käytetään Chromen selaimessa.
- Thread pool on se osa, joka hoitaa tiedostojen syöttö- ja tuotto-operaatiot. Kaikki järjestelmän blokkaavat kutsut suoritetaan tässä.
- Event loop-kirjasto. (13, s. 2.)



KUVA 5. Node.js:n arkkitehtuuri (13, s. 2.)

Noden peruskirjasto on kirjoitettu JavaScriptillä ja muut osaat C:llä.

4.1 V8-moottori

V8-moottori on avoimen lähdekoodin projekti Googlelta ja Google Chromen selaimen ydin. Ensimmäinen julkinen versio ilmestyi syyskuussa 2008, samaan aikaan kun Chrome-selaimen ensimmäinen versio julkistettiin. (14.)

V8 oli suuri askel eteenpäin selaimien suorituskyvyn tehostamisen kannalta ja nosti selaimien teknologiaa ihan uudelle tasolle. Moottori on kirjoitettu pääosin C++-ohjelmointikielellä. (14.)

4.2 Thread pool

Thread pool (libeio) on monipuolinen asynkroninen I/O kirjasto C:lle. Se on mallinnettu samanlaisella tyylillä kuin libev. Se sisältää ominaisuuksia kuten asynkroninen lukeminen, kirjoittaminen, avaaminen, sulkeminen, stat, unlink, fdatasync, mknod, readdir jne. (15.)

Se on täysin riippumaton tapahtumakirjasto ja sen pystyy integroimaan mihin tahansa tapahtumakirjastoon. Se on erittäin siirrettävä ja käyttää pelkästään POSIX-säikeistä. (15.)

4.3 Event pool

Event pool (libev) on tehokas tapahtumasilmukka C:lle. Libev tukee I/O:ita, ajastimia, signaaleita, prosessin tilamuutoksia ja muita tapahtumatyyppejä. Se sisältää nopean paikallisen rajapinnan ja libevent-emuloinnin ja tukee ohjelmia kirjoitettuna käyttäen libevent APIa. Erot libeventiin ovat korkeampi nopeus, yksinkertaisempi malli, enemmän ominaisuuksia ja vähemmän muistin käyttöä. Libev tukee epoll-kutsuja, kqueue-toimintoa, Solariksen tapahtumaportteja ja pollausta. (16.)

4.4 Node Package Manager

Node Package Manager (npm) on oletuspakettimanageri, joka on kehitetty Node.js:lle. Se ilmestyi ensimmäistä kertaa Node.js:n versiossa 0.6.3, ja se ny-

kyään tulee sen kanssa asennuksen yhteydessä. Oletuspakettimanageria suoritetaan komentorivin kautta ja se hoitaa tarvittavat riippuvuudet kehitettävää sovellusta varten. Npm on kehitetty täysin JavaScript-ohjelmointikielellä. (17.)

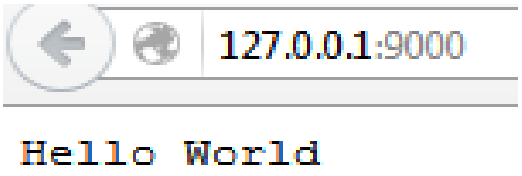
4.5 Node.js-palvelimen käynnistys

Node.js:n voi hakea kehitysalustan kotisivulta https://nodejs.org/en/. Kehitysalusta on saatavilla monelle käyttöjärjestelmälle, kuten Windows, Linux ja OS X. Node.js:ää käytetään komentokehotteella asennuksen jälkeen. Kuvan 6 koodi on esimerkki Node.js:n käytöstä. (13, s. 3.)

```
var http = require('http');
http.createServer(function(req, res) {
    res.writeHead(200, {
        'Content-Type': 'text/plain'
    });
    res.end('Hello World\n');
}).listen(9000, '127.0.0.1');
console.log('Server running at http://127.0.0.1:9000/');
```

KUVA 6. Node.js-esimerkki

Koodin ajamiseen käytetään node-komentoa kirjoittamalla komentokehotteeseen ”node ./server.js”, jolloin käynnistyy palvelin (kuva 7) osoitteessa http://127.0.0.1:9000.



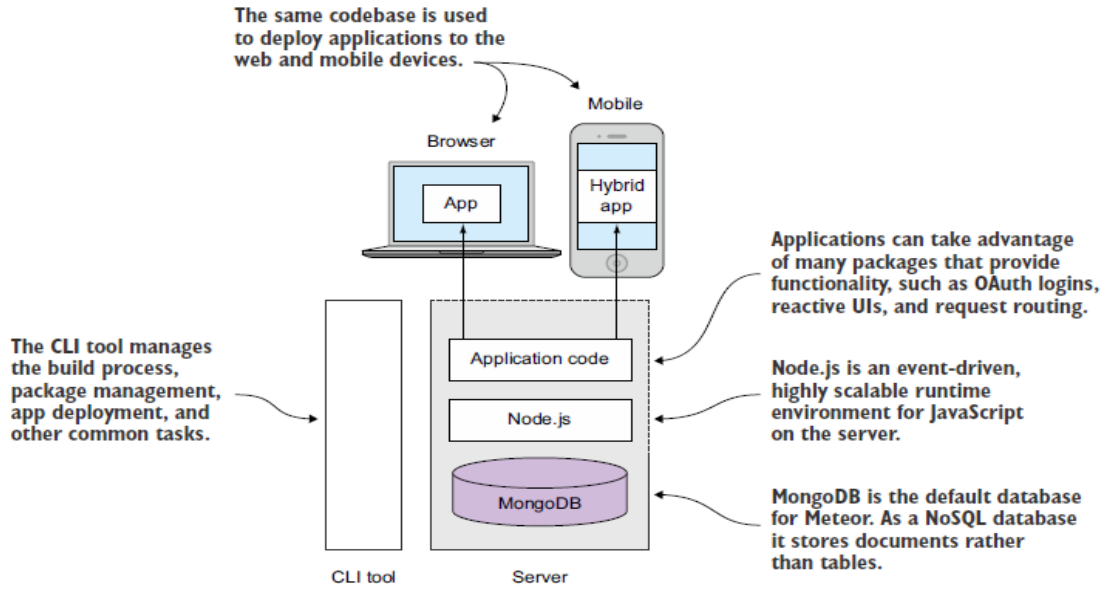
KUVA 7. Esimerkin tulos

5 METEOR-SOVELLUSKEHYS

Meteor on avoimen lähdekoodin sovelluskehys, joka on tarkoitettu dynaamisten web-sovelluksien kehittämiseen täysin JavaScriptillä. Meteor yhdistää ja välittää kaikki tarvittavat osat yhteen alustaan. Se koostuu Node.js:stä, MongoDB:stä, sovelluksen koodista ja voimakkaasta komentoliittymästä, joka yhdistää npm:n ja maken. Sellaisenaan alusta on enemmän kuin palvelimen prosesseja ja kirjastoja. (18, s. 5.)

Meteor Development Group (MDG) julkaisi joulukuussa vuonna 2011 Skybreakin ensimmäisen esikatseluversion, joka nimettiin pian Meteoriksi. Kahdeksan kuukautta myöhemmin projekti sai 11,2 miljoonan dollarin rahoituksen teollisuuden edustajilta. GitHubissa Meteor on pysynyt suosituimpien projektien joukossa. (18, s. 5.)

Meteor on täydellinen paketti web-sovelluksien kehittämiseen (kuva 8).



KUVA 8. Meteorin rakenne (18, s. 6)

Meteor käyttää palvelimessa Node.js:ää, jolla on sama tarkoitus kuin Apache-palvelimella LAMP-ohjelmassa. Sovelluksen data tallennetaan MongoDB:hen, joka on dokumenttipohjainen NoSQL-tietokanta. (18, s. 6.)

Meteor tällä hetkellä tukee virallisesti pelkästään MongoDB-tietokantaa, mutta on suunnitelmia tulevaisuudessa tukea muita tietokantatyyppejä. MongoDB tarjoaa JavaScript APIn, joka mahdollistaa pääsyn tallennettuun dataan dokumentti- tai objektimuodossa. (18, s. 6.)

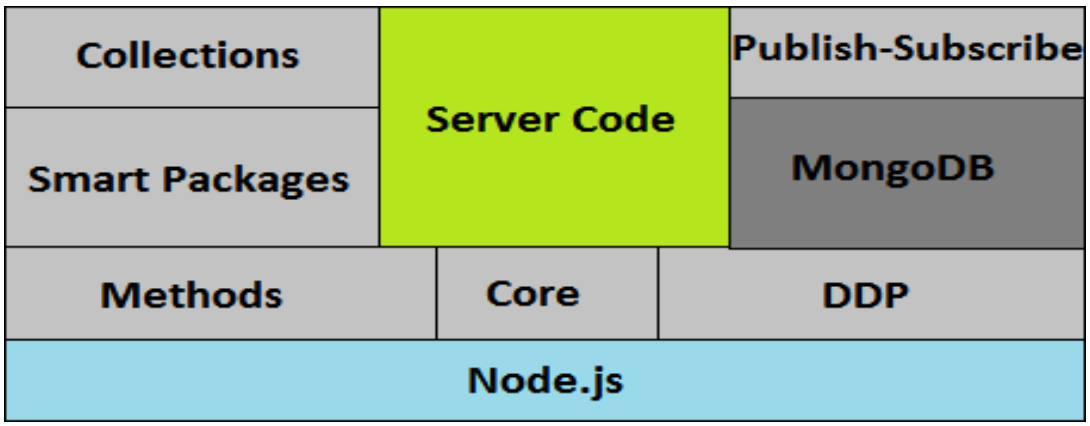
Kaikki web-sovelluksen kehittämiseen tarvittavat sovellukset ja kirjastot on yhdistetty pieniin paketteihin, jotta kehittäjät pääsevät aloittamaan työn välittömästi. Sellaisia paketteja ovat esimerkiksi reaktiivinen UI-kirjasto (Blaze), käyttäjätilin hallinta (accounts) ja kirjasto reaktiivista ohjelmointia varten (Tracker). (18, s. 6.)

Meteorin komentoliittymällä kehittäjät asentavat nopeasti kehitysympäristön eikä heidän ole tarpeen tietää, miten asennetaan tai säädellään jokin palvelimen ohjelmistoa (18, s. 6).

5.1 Arkkitehtuuri

5.1.1 Palvelinpuoli

Meteor koostuu palvelinpuolelta kuvan 9 komponenteista.

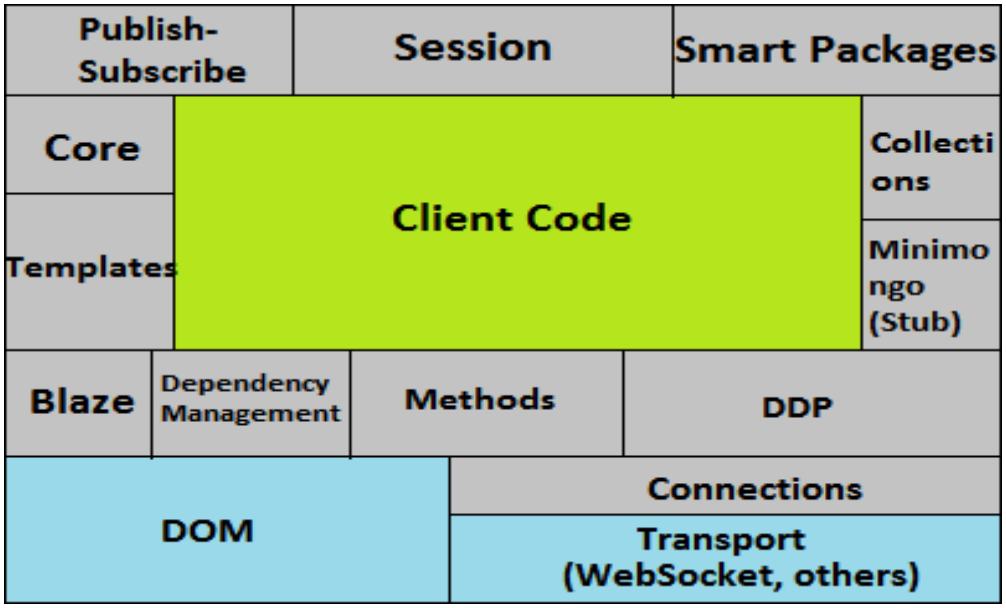


KUVA 9. Palvelinpuolen arkkitehtuuri

- Collections eli kokoelmat ovat Meteorin tapa tallentaa pysyvää dataa ja niihin on pääsy sekä palvelinpuolelta että asiakaspuolelta.
- Meteorin palvelinpuoli perustuu Node.js-kehitysalustaan.
- Publish ja Subscribe ovat funktioita, joita käytetään datan välittämiseen palvelimelta asiakaspuolelle.
- Coressa on Meteorin ytimen paketit.
- Smart Packages oli Meteorin vanhoissa versiossa pakettiasentaja. Nykyään paketteja asennetaan meteor-komennolla komentokehotteella.
- Meteorissa Methods eli metodeja käytetään varmistamaan, että nykyisellä käyttäjällä on oikeus tehdä muutoksia tietokannan tietoihin.
- DDP on protokolla tiedon siirtymistä varten.
- Meteor käyttää virallisesti MongoDB:tä. (19, linkit Develop -> Documentation)

5.1.2 Asiakaspuoli

Meteor koostuu asiakaspuolelta seuraavista komponenteista (kuva 10).



KUVA 10. Asiakaspuolen arkkitehtuuri

- Templatea eli mallia käytetään yhteyden luomiseen JavaScriptin ja käyttöliittymän välillä.
- Blaze on kirjasto dynaamista käyttöliittymää varten.
- Session liittyy lyhytaikaiseen datan talletukseen.
- Minimongo on pakollinen paketti, kun käytetään MongoDB:tä asiakaspuolella. (19, linkit Develop -> Documentation.)

5.2 Paketit ja teknologiat

Meteor sisältää kahden tyypin paketteja: 1) oletuspaketit, jotka tulevat automaattisesti lisättyä projektia luotaessa, ja 2) yhteisön tekemiä paketteja, joita voidaan lisätä komentorivillä.

5.2.1 Blaze

Blaze on tehokas kirjasto käyttöliittymän päivitystä varten. Blaze on helpompi käyttää kuin esimerkiksi Angular, Backbone, Ember, Polymer tai Knockout. Kirjasto luotiin helpottamaan käyttöliittymän kehittämistä. (19, linkit Guide -> Blaze.)

Blazella kirjoitetaan tavallista HTML-syntaksia (kuva 11).

```
<div class="friendList">
  <ul>
    {{#each friends}}
      <li class="{{#if selected}}selected{{/if}}">
        {{firstName}} {{lastName}}
      </li>
    {{/each}}
  </ul>
</div>
```

KUVA 11. Esimerkki Blaze-koodista

Käyttöliittymä päivittyy automaattisesti aina, kun on tapahtunut muutoksia käytettävään dataan. HTML-syntaksiin on lisätty aaltosulkeita, joilla välitetään dataa käyttöliittymälle käyttämällä JavaScriptiä toisessa tiedostossa. Meteor tunnistaa muutoksen datassa käyttämällä Trackeria. (19, linkit Guide -> Blaze.)

5.2.2 Tracker

Tracker-paketti tarjoaa toiminnallisen reaktiivisen ohjelmoinnin perusteet. Tracker on rajapinta, jossa reaktiivisen datan lähteet kuten tietokanta ovat yhteydessä reaktiivisen datan käyttäjän kanssa. Trackerin rajapinta on käytännössä yksinkertainen, mikä antaa mahdollisuuden kehittäjille lisätä omiin paketteihin reaktiivisuutta. (19, linkit Develop -> Documentation.)

5.2.3 Livequery

Livequery on live-tietokannan liittymä. Livequery tunnistaa kaikki datan muutokset tietokannassa ja välittää muutokset asiakkaiden käyttöliittymän osiin. (19, linkit Guide -> Deployment and Monitoring.)

5.2.4 DDP

DDP (Distributed Data Protocol) on yksikertainen protokolla datan noutamiseen palvelinpuolelta. Se tekee suoria päivityksiä silloin, kun dataan on tullut muutoksia. DDP kuten REST-protokolla on yksikertainen käytännönläheinen tapa tehdä rajapinta. Se perustuu web-socketiin, joka antaa mahdollisuuden käyttää suoria päivityksiä toisin kuin REST-protokolla. (19, linkit Guide -> Publications and subscriptions.)

Meteorin sovelluksissa asiakkaat ovat yhteydessä palvelimien kanssa DPP:llä.

5.2.5 Isobuild

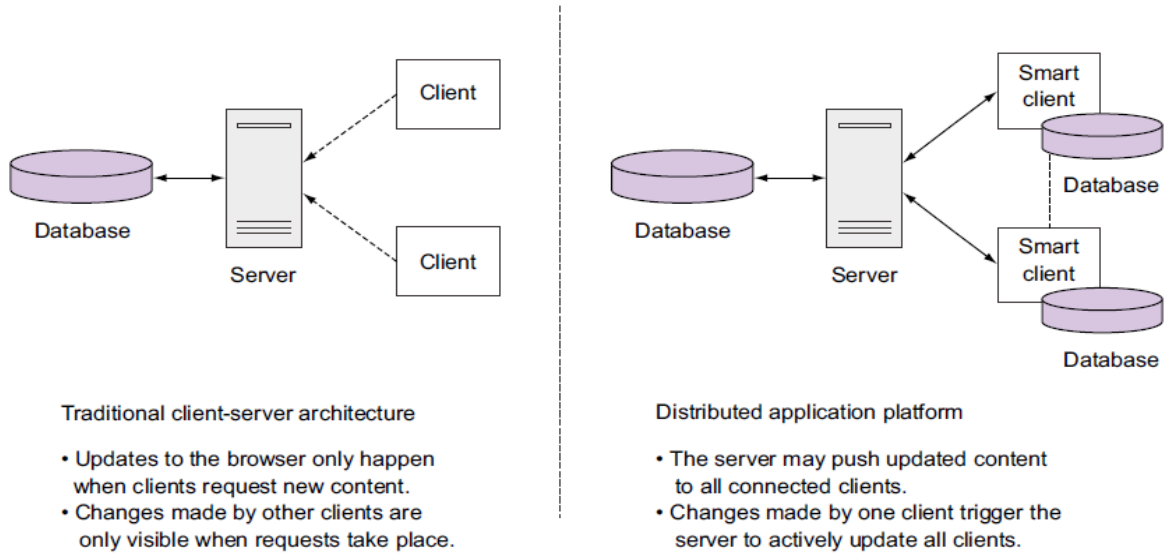
Isobuild on täydellinen työkaluketju web-sovelluksien kehittämiseen. Silloin, kun halutaan ajaa web-sovellus, Isobuild kerää kaikki lähdetiedostot ja kaikki käytetyt paketit ja prosessoi ne sopivan rakennusvaiheen mukaisesti. Isobuildilla voidaan tulostaa monenlaisia ajettavia sovelluksia, kuten mobiilisovellus iOSille tai Androidille. (19, linkit Guide -> Build system.)

5.3 MongoDB

MongoDB on avoimen lähdekoodin tietokanta, joka käyttää dokumenttisuuntautunutta datamallia. 2000-luvun puolivälissä ilmestyi NoSQL-kategorian tietokantoja ja yksi niistä on MongoDB, joka on suosituin NoSQL-tietokanta tällä hetkellä. MongoDB on rakennettu arkkitehtuuriin, joka sisältää kokoelmia ja dokumentteja. Dokumentit koostuvat avain-arvo-pareista ja ne ovat MongoDB:n perusyksikkö. Kokoelmat sisältävät dokumentteja ja funktioita. (20.)

Kuten muut NoSQL-tietokannat MongoDB tukee dynaamista mallin suunnittelua, mikä antaa mahdollisuuden tehdä dokumentteihin erilaisia kenttiä ja rakenteita. Tietokanta käyttää JSONia binaarisessa muodossa. (20.)

Meteor käyttää pakettia minimongo, joka on MongoDB:n APIn uudelleenimplementaatio (kuva 12).



KUVA 12. Tiedon siirto tietokannasta Meteorissa (18, s. 10)

Minimongon ja reaktiivisen kirjastoiden avulla sovelluksesta on aina suora yhteys palvelimeen, joka mahdollistaa tiedon päivityksen välittömästi (19, linkit Guide -> Collections and Schema).

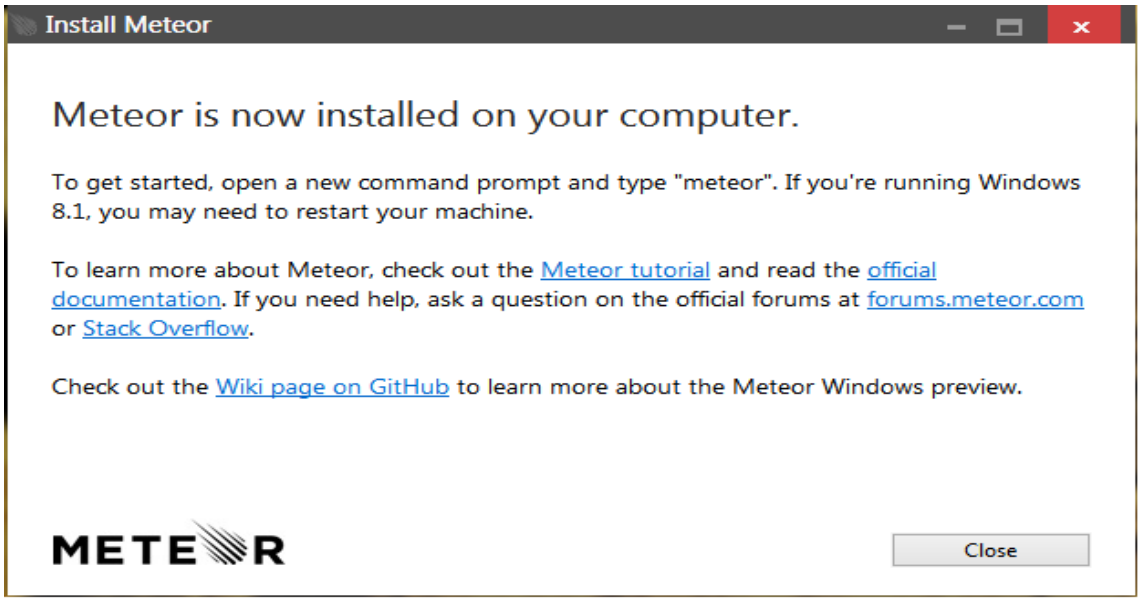
6 WEB-SOVELLUKSEN TOTEUTUS

6.1 Prototyypin tavoite

Prototyypin tavoite oli kehittää virtuaaliympäristö, jossa opiskelijat voivat harjoitella osakkeisiin sijoittamista ilman negatiivisia seuraamuksia. Osakkeiden ja kurssien tiedot haetaan käyttämällä yhden taloussivun rajapintaa. Web-sovellusta on tarkoitus käyttää opetuksessa apuvälineenä, koska se sisältää myös oppimismateriaalia aiheista, kuten säästämisestä, pörssin toiminnasta ja sijoitusmuodoista.

6.2 Meteorin asennus

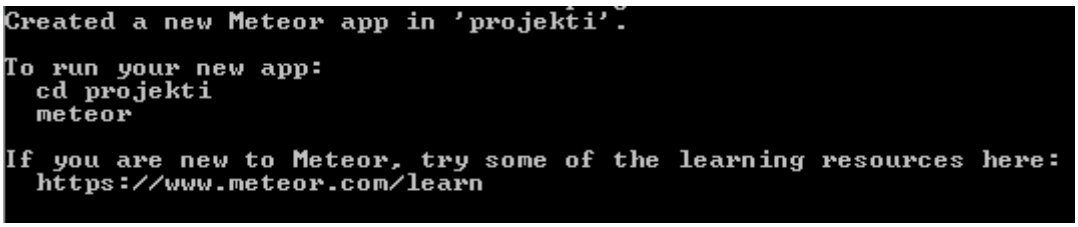
Aloitettiin hakemalla kehitysalustan viimeinen versio verkkosivulta https://www.meteor.com. Meteorin voi asentaa useampiin käyttöjärjestelmiin kuten OS X, Windows ja Linux, mutta tässä tapauksessa se asennettiin Windows-käyttöjärjestelmään (kuva 13).



KUVA 13. Meteorin asennus

6.3 Projektin luonti

Windows-käyttöjärjestelmässä uusi projekti luotiin avaamalla käyttöjärjestelmän komentokehote, johon syötetiin komento ”meteor create projekti” (kuva 14).



KUVA 14. Projektin luonti

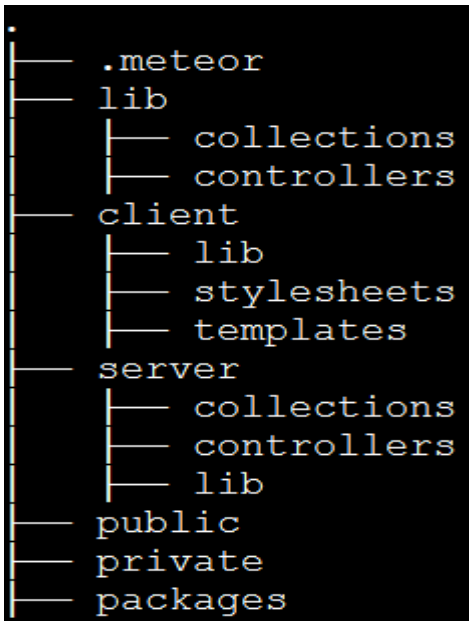
Web-sovelluksen voi käynnistää menemällä komentokehotteella luotuun projektin kansioon syöttämällä komennon ”meteor”.

Käynnissä olevaa web-sovellusta pääsee kokeilemaan selaimella osoitteesta http://localhost:3000.

6.4 Web-sovelluksen rakenne

Meteor ei pakota seuraamaan tiettyä hakemistorakennetta projektille, mutta joillakin kansioilla on erityinen tarkoitus, kuten server- ja client-kansioilla. Kansioiden nimillä voidaan asettaa rajoituksia sovellukselle, kuten erottaa asiakaspuoli palvelimenpuolesta tai projektin resurssit koodista.

Projektin luomisen jälkeen lisättiin kuvan 13 mukaisesti rakenne web-sovellukselle.



KUVA 13. Projektin rakenne

- .meteor-kansio on olemassa kaikissa Meteorin projekteissa. Se sisältää sovelluskehyksen tarvitsemat tiedostot projektin ajamista varten.
- client-kansion sisältö suoritetaan pelkästään asiakaspuolella.
- server-kansion sisältö suoritetaan pelkästään palvelinpuolella.
- lib-kansion sisältö suoritetaan molemmissa ympäristössä.
- public-kansioon on tarkoitus tallentaa tiedostoja kuten kuvia ja videoita asiakaspuolelle.
- private-kansioon on pääsyoikeus vain palvelimelta.
- package-kansioon on tarkoitus lisätä omia paketteja.

Kolmelle kansiolle server, client ja lib lisättiin alikansioita:

- collections-kansioon on tarkoitus lisätä MongoDB:n tauluja. Ne voivat olla pelkästään palvelimenpuolella tai molemmissa.
- controllers-kansio on lisätty Iron Router-paketille.
- lib-kansio on tarkoitettu JavaScriptin kirjastoille, joita ei voi lisätä komentorivillä.
- stylesheets-kansioon tallennetaan pääosiin CSS-tiedostoja.

- templates-kansio on tarkoitettu HTML-malleille.

6.5 Käyttöliittymän päämalli

Käyttöliittymän kehittäminen aloitettiin asentamalla Bootstrap-paketti komennolla ” meteor add twbs:bootstrap” ja luomalla web-sivulle oletusmalli kaikille sivuille. Polulle ”templates/layouts” lisättiin master_layout-kansio. Kansion sisälle luottiin ”master_layout.html”, johon lisättiin päämalli (kuva 14).

```
<template name="MasterLayout">
</template>
```

KUVA 14. Templaten määritys

Päämalli koostuu kolmesta osata:

1. Ylätunniste, jossa on sivun nimi ja sivun vasemmalle menu-painike ja oikealle asetukset-nappi
2. Sivun varsinainen sisältö, joka on ylätunnisteen ja alatunnisteen väliä
3. Alatunniste, jossa on työntilaajan nimi ja nappi siirtää sivu ylöspäin.

Käyttämällä Bootstrapin komponentteja lisättiin ylätunniste eli navigaatiopalkki (kuva 15) web-sovellukselle.

```
<nav class="navbar navbar-default navbar-fixed-top navbar-fixed-width">
  <div class="container">
    <a class="  navbar-brand" href="#">Sivun nimi</a>
    <div>
      <ul class="nav navbar-nav navbar-left">
        <li>
          <div>
            <a href="#menu" class="btn navbar-btn"><span class='glyphicon glyphicon-align-justify'></span>
        </a>
          </div>
        </li>
      </ul>
      <ul class="nav navbar-nav navbar-right">
        <li>
          <div <a href="#" id="settings" class="btn navbar-btn" <span class='glyphicon glyphicon-cog'></span>
            </a>
          </div>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

KUVA 15. Ylätunniste

Ylätunnisteen alle tulee sivun varsinainen sisältö ja sitä varten lisättiin”{{> yield}}”. Se on yksi Iron Routerin ominaisuuksista, jolla lisätään dynaamisesti sisältöä sen mukaan, mikä sivu on avattuna.

```
<div class="container">
  {{> yield}}
</div>
```

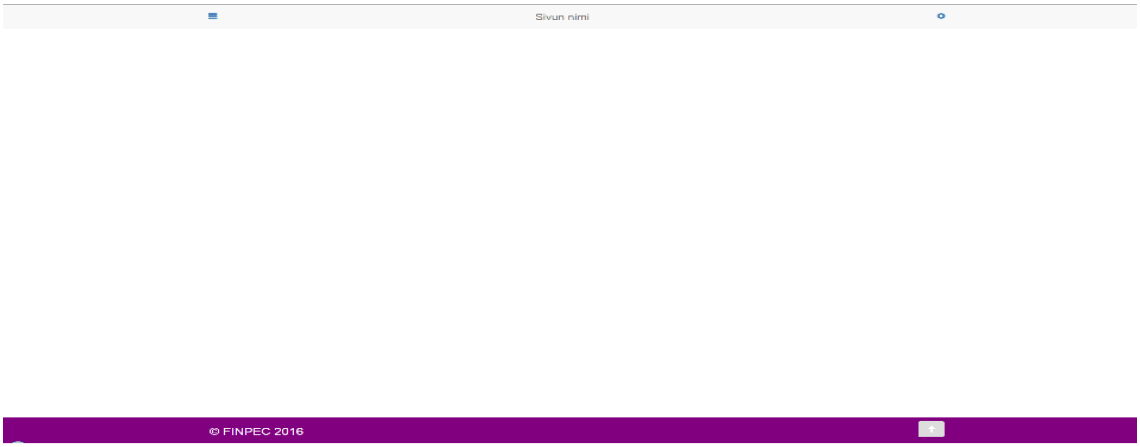
KUVA 15. Iron Routerin ominaisuus

Seuraavaksi lisättiin alatunniste samalla periaatteella kuin ylätunniste (kuva 16).

```
<div class="navbar navbar-custom navbar-fixed-bottom">
  <div class="container">
    <div class="col-md-4 ">
      <p class=""> © FINPEC 2016</p>
    </div>
    <div class="pull-right">
      <ul class="nav navbar-nav">
        <li>
          <button id="scroll-up" class="btn navbar-btn"><span class='glyphicon glyphicon-arrow-up'></span></button>
        </li>
      </ul>
    </div>
  </div>
</div>
```

KUVA 16. Alatunniste

Kaikki osat yhdistettynä saatiin päämalli valmiiksi (kuva 17).



KUVA 17. Päämalli toteutettuna

Toteutettu päämalli asetettiin seuraavassa luvussa oletusmalliksi kaikille sivuille.

6.6 Navigaatio

Siirtyminen yhdestä sivusta toiseen web-sovelluksessa yksi tärkeimmistä ominaisuuksista heti alussa, ja sitä varten lisättiin komentorivillä projektiin paketti nimellä Iron Router ”meteor add iron:router”. Iron Routerilla saadaan web-sovellukseen dynaaminen siirto yhdestä sivusta toiseen.

Paketin lisäyksen jälkeen luottiin JavaScript-tiedosto kansioon ”lib” nimellä ”routes”, johon määritettiin oletusmalli ja reitit web-sovellukselle. Configure-funktiolla (kuva 18) asetettiin kolmelle tapaukselle oma HMTL-malli.

```
Router.configure({
  layoutTemplate: 'MasterLayout',
  loadingTemplate: 'Loading',
  notFoundTemplate: 'NotFound'
});
```

KUVA 18. Routerin konfiguraatio

Iron Routerilla voidaan määrittää koko web-sovellukselle yksi malli, joka on käytössä jokaisessa sivussa, ja sen sisälle sivun reitin mukaan lisätään tarvittava sisältö. ”MasterLayout” on tässä tapauksessa oletusmallin nimi, joka lisättiin aikaisemmin. loadingTemplate ja notFoundTemplate eivät ole pakollisia määrittää, mutta niistä on hyötyä ison datan latauksessa tai väärän reitin tapauksessa.

Reittejä voidaan lisätä router-funktiolla (kuva 19), jolle annetaan reitin nimi, ohjaimen nimi, toiminta ja tieto, missä tapahtuu siirtyminen.

```
Router.route('/', {
  name: 'main',
  controller: 'MainController',
  action: 'action',
  where: 'client'
});

Router.route('login', {
  name: 'login',
  controller: 'LoginController',
  action: 'action',
  where: 'client'
});
```

KUVA 19. Reittien määritys

Web-sivulle määritettiin kaksia reittiä ”/” ja ”login”. Ensimmäisen reitin on web-sovelluksen pääsivu ja toinen reitin on kirjautumissivu.

Luottiin lib-kansion sisälle controllers- niminen kansio, johon lisätiin ohjaimet eli laajennukset reitittimille main ja login. Reitittimiä voidaan laajentaa monella eri funktiolla (kuva 20).

```
MainController = RouteController.extend({
  layoutTemplate: 'MasterLayout',

  subscriptions: function() {
  },

  data: function() {
  },

  action: function() {
    if (Meteor.user()) {
        this.render('Main');
    } else {
      this.render('Login');
    }
  }
});
```

KUVA 20. main_controller.js-tiedoston sisältö

MainControlleriin määritettiin subscriptions-, data- ja action-funktiot. Action-funktiossa on lisätty tarkistus, onko käyttäjä siirron aikana kirjautunut web-

sovellukseen. Kirjautunut käyttäjä siirretään normaalisti pääsivuun toisin kuin kirjautumaton käyttäjä siirretään kirjautumissivulle. Main-sivussa on käytössä pääoletusmalli ”MasterLayout” kuten toisissakin sivuissa.

6.7 Kirjautuminen

Kirjautumista varten lisättiin seuraavat paketit web-sovellukseen:

- accounts-base ja accounts-password käyttäjätilijärjestelmän luontia varten
- aldeed:autoform ja aldeed:simple-schema helpottamaan lomakkeiden luontia
- numtel:mysql yhteyden muodostamista varten MySQL-tietokantaan

6.7.1 Näkymä

Autoform-paketilla voidaan lisätä UI-komponentteja ja helper-funktioita eli autta-ja-funktioita helpottamaan peruslomakkeiden luontia. Lomakkeilla on automaattinen tapahtumien lisäys ja päivitys operaatoreille ja automaattinen reaktiivinen validointi.

Kirjautumislomake saatiin lisättyä käyttämällä Autoformia (kuva 21).

```
<template name="Login">
  <div class="container">
  <div class="row">
    <div class="col-md-2 col-md-offset-5">
      {{#autoForm id="loginForm" schema=loginFormSchema}}
      <fieldset>
        <legend>{{_ 'loginTitle'}}</legend>
        {{#each afFieldNames}}
           {{> afQuickField name=this.name options=afOptionsFromSchema}}
         {{/each}}
        <div>
          <button id="login-button" type="submit" class="btn btn-primary">{{_ 'loginButton'}}</button>
        </div>
      </fieldset>
      {{/autoForm}}
    </div>
  </div>
</div>
</template>
```

KUVA 21. Lomake toteutettu Autoformilla

Autoform luo lomakkeen tarvittavat kentät syöttämällä sille sopivan rakenteen, joka voi olla MongoDB:n taulu tai Schema. Määritettiin Schema käyttämällä simple-schema-pakettia (kuva 22).

```
LoginSchema = new SimpleSchema({
  email: {
    type: String,
    label: "Username",
    regEx: SimpleSchema.RegEx.Email
  },
  password: {
    type: String,
    autoform: {
      afFieldInput: {
        type: 'password'
      }
    }
  }
});
```

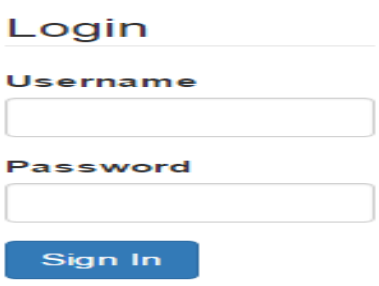
KUVA 22. Scheman määritys

Kirjautumista varten tarvittiin pelkästään email- ja password-kentät. Scheman välitys Autoformin tehtiin luomalla auttaja eli helper-funktio (kuva 23), joka palauttaa Autoformille Scheman tiedot lomakkeen kenttien luontia varten.

```
Template.Login.helpers({
  loginFormSchema: function() {
    return LoginSchema;
  }
});
```

KUVA 23. Helper-funktio

Autoform luo lomakkeen sivulle käyttämällä Schemasta saatuja tietoja (kuva 24).



KUVA 24. Kirjautumislomake

6.7.2 Lomakkeen lähettäminen

Painamalla kirjautumisnappia ei tapahdu mitään, koska lomaketta ei ole liitetty mihinkään funktioon. Lähettämisen tapahtumaa päästiin käsittelemään lisäämällä Autoformiin hook-funktio (kuva 25).

```
AutoForm.hooks({
  loginForm: {
    onSubmit: function(insertDoc, updateDoc, currentDoc) {
```

KUVA 25. Hook-funktio

onSubmit-funktiota kutsutaan aina, kun painetaan kirjautumisnappia, ja sillä on parametreina lomakkeelle syötetyt uudet arvot, joilla tässä tapauksessa tarkistetaan, vastaavatko ne ketään käyttäjää. Funktion sisällä kutsutaan palvelimenpuolelta metodia ”checkLoginInformation” (kuva 26), jossa tarkistetaan syötetyt uudet arvot ja annetaan lupa kirjautua web-sovellukseen tai ilmoitetaan epäonnistumisesta.

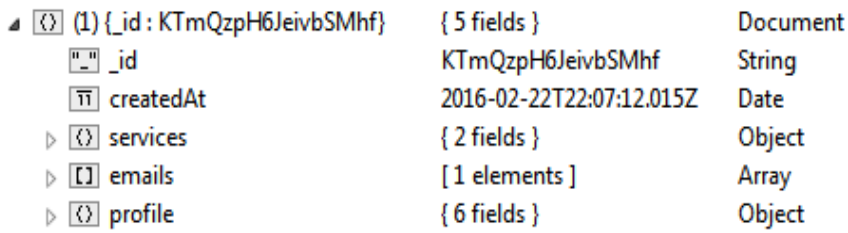
```
Meteor.call("checkLoginInformation", insertDoc, function(err, result)
```

KUVA 26. Metodin kutsu

6.7.3 Käyttäjätilit

Accounts-paketit lisäävät web-sovellukseen täydellisen käyttäjätilijärjestelmän, jolla saa vähällä koodilla ison määrän ominaisuuksia kuten sisäänkirjautumisen, uloskirjautumisen, tilin luonnin, sähköpostin validoinnin ja kirjautumisen ulkopuolisella palvelulla.

Käyttäjät tallennetaan automaattisesti Users-kokoelmaan MongoDB:ssä. Sen rakenne (kuva 27) on sama kaikissa tapauksissa ja muokkauksia pystyy tekemään pelkästään profile-kentän sisälle.



KUVA 27. User-kokoelman sisältö

Käyttäjän luonti tapahtuu funktiolla createUser(username/email,password). Funktiota pystyy kutsumaan sekä asiakaspuolelta että palvelinpuolelta.

Kirjautuminen tapahtuu funktiolla loginWithPassword(username/email,password). Funktiota pystyy kutsumaan pelkästään asiakaspuolelta.

6.7.4 Tietokantayhteyden muodostaminen MySQL:ään

Käyttäjien tiedot ovat alun perin MySQL-tietokannassa. Sen vuoksi piti käyttää yhteisön tekemää pakettia, jolla varmistetaan, vastaavatko lomakkeessa syötetyt arvot jotain käyttäjää MySQL-tietokanassa, ennen käyttäjän luontia User-kokoelmaan MongoDB:ssä.

Yhteyden luonti MySQL:ään tapahtuu funktiolla LiveMysql (kuva 28) ja kyselyitä voi tehdä funktiolla ”liveDb.db.query”.

```
var liveDb = new LiveMysql({
  host: 'localhost',
  port: 3407,
  user: 'root',
  password: '',
  database: 'tietokannan nimi'
});
```

KUVA 28. Yhteyden luonti

6.8 Monikielisyys

Web-sovelluksella oli vaatimus tukea kolme eri kieltä: englanti, suomi ja ruotsi. Meteorissa kielenvaihto tapahtuu käyttämällä yhteisön tekemää pakettia ”tap-i18n”. Jokaiselle kielelle piti pelkästään luoda ”.json”-tyyppinen tiedosto, jossa on jokaiselle avainsanalle määritetty käännös (kuva 29).

```
"logout" : "Logout",
"loginTitle" : "Login",
"loginButton" : "Sign In",
"saveButton" : "Save",
"language" : "Language",
"changeLanguage" : "Change language",
"selectLanguage" : "Select language",
"english" : "English",
"finnish" : "Finnish",
"swedish" : "Swedish",
```

```
"logout" : "Kirjaudu ulos",
"loginTitle" : "Kirjautuminen",
"loginButton" : "Kirjaudu",
"saveButton" : "Tallennus",
"language" : "Kieli",
"changeLanguage" : "Vaihda kieli",
"selectLanguage" : "Valitse kieli",
"english" : "Englanti",
"finnish" : "Suomi",
"swedish" : "Ruotsi",
```

KUVA 29. en.i18n.json- ja fi.i18n.json-tiedostojen sisältö

Ylätunnisteen asetukset-nappiin lisättiin valinta kielen vaihtoon (kuva 30).

```
<li class="dropdown-submenu">
  <a tabindex="0">{{_ 'changeLanguage'}}
  </a>
  <ul class="dropdown-menu">
    <li class="dropdown-header">{{_ 'language'}}</li>
    <li>
      <a name='en' tabindex="0">{{_ 'english'}}</a>
    </li>
    <li>
      <a name='fi' tabindex="0">{{_ 'finnish'}}</a>
    </li>
    <li>
      <a name='sv' tabindex="0">{{_ 'swedish'}}</a>
    </li>
  </ul>
</li>
```

KUVA 30. Asetukset-nappi

Kaksoissulkujen sisällä on avainsana kielitiedostoille. Näkyvä kieli riippuu valitusta kielestä, jonka voi vaihtaa TAPi18n.setLanguage(language)-funktiolla (kuva 31).

```
$(".dropdown-menu li a").click(function() {
  console.log("Selected Option:" + $(this).text());

  if ($(this).attr("name") == "en") {
    TAPi18n.setLanguage("en");
  } else if ($(this).attr("name") == "fi") {
    TAPi18n.setLanguage("fi");
  } else if ($(this).attr("name") == "sv") {
    TAPi18n.setLanguage("sv");
  }
});
```

KUVA 31. Kielenvaihto painaluksen jälkeen

6.9 Tiedoston siirto

Tiedoston lähetys Meteorissa tapahtuu joko meteor-uploadsilla tai CollectionFS-paketilla. Tässä web-sovelluksessa päätettiin käyttää meteor-uploads-pakettia, koska se on helpompi toteuttaa ja tarvittiin tiedoston siirtoa palvelimeen pelkästään profiilin kuvaa varten.

Asennettiin meteor-uploads-paketti, joka koostuu kahdesta pakettista tomi:upload-server ja tomi:upload-jquery. Paketti vaatii palvelimenpuolelta asetusta (kuva 32). Web-sovelluksen käynnistyessä asetetaan paketille tallennuskansio ja hyväksyttävät tiedostotyypit, jolla estetään väärien tiedostojen tallentamista.

```
Meteor.startup(function() {
  UploadServer.init({
    tmpDir: application_root + '/.private/tmp',
    uploadDir: application_root + '/.private/',
    checkCreateDirectories: true,
    mimeTypes: {
      "jpeg": "image/jpeg",
      "jpg": "image/jpeg",
      "png": "image/png"
    },
    acceptFileTypes: /(.|\/)(gif|jpe?g|png)$/i
  })
});
```

KUVA 32. Paketin asetus

Paketti sisältää valmiin käyttöliittymän, joka on toteutettu Bootstrapilla, jonka voi lisätä helposti HTML:ään lauseella ”{{> upload_bootstrap }}” (kuva 33).

KUVA 33. Paketin käyttöliittymä

Kuvan tallentumisen jälkeen päivitetään käyttäjän kenttää ”profile.image” tiedolla, johon tulee talletetun kuvan osoite (kuva 34).

```
  saveImage: function(fileData) {
    if (Meteor.userId()) {
      Meteor.users.update(
        Meteor.userId(), {
          $set: {
            "profile.image": fileData.url,
          }
        });
      return true;
    } else {
      return null;
    }
  }
});
```

KUVA 34. Käyttäjän kuva-kentän päivittäminen

Käyttäjän profiilin kuva esitetään web-sovelluksessa käyttämällä img-elementtiä (kuva 35), jolle on määritetty helper-funktio ”{{userImage}}”.

```
<img id="image" src="{{userImage}}" class="img-responsive img-circle" alt="No image">
```

KUVA 35. Kuvan määritys

Funktio palauttaa käyttäjän profiilin kuvan osoitteen palvelinpuolelta (kuva 36).

```
userImage: function() {
  return Meteor.user().profile.image;
}
```

KUVA 36. Kuvan osoitteen palautus tietokannasta

6.10 Tietokannan turvallisuus

User-kokoelmaa pystyy muokkaamaan selaimen konsolista esimerkiksi komennolla ”Meteor.users.insert({emails: [user@email.com'] });”, joka lisää kokoelmaan uuden dokumentin. Konsolia voi käyttää kuka tahansa web-sovelluksen vierailija, joka voi lisätä User-kokoelmaan satunnaista tietoa ja vaikuttaa web-sovellukseen toimintaan.

Tietokannan muokkaamisen voi estää käyttämällä Meteorin allow- ja deny-komentoja, joilla pystyy hyväksymään tai hylkäämään konsolista tai asiakaspuolelta syötettyjä komentoja joidenkin ehtojen mukaan. User-kokoelman muokkaaminen estettiin deny-komennolla (kuva 37).

```
Meteor.users.deny({
  insert: function(userId, doc) {
    return true;
  },
  update: function(userId, doc, fields, modifier) {
    return true;
  },
  remove: function(userId, doc) {
    return true;
  }
});
```

KUVA 37. deny-funktion käyttö

Estot tehtiin palvelinpuolella ja asetettiin kokonaan kielto muokkauksiin konsolilta tai asiakaspuolelta. Muokkaukset tapahtuvat nyt pelkästään palvelinpuolelta eli palvelimen metodeilla. Samalla tavalla pystyy lisäämään ehtoja toisiin kokoelmiin suojaamalla tietokantaa.

6.11 Osakkeet

Web-sovelluksessa haluttiin ensiksi esittää luettelo Helsingin pörssin osakkeista ja toiseksi pystyä hakemaan lisätietoja valitusta osakkeesta, kuten yhden vuoden päätöskurssit esitettynä viivakaaviolla ja edellisen päivän päätöskurssi.

6.11.1 Lista osakkeista

Osakkeiden luetteloa varten asennettiin paketti meteor-pages, joka ottaa vastaan Collection-tyyppisen muuttujan ja luo siitä listan, joka sisältää sivunumeroinnin. Lisättiin routes.js-tiedostoon paketin rakentaja (kuva 38), jolle annettiin parametrina Stocks, joka on tässä tapauksessa Collection-tyyppisen muuttujan nimi, johon on tallennettu osakkeiden nimet. Toinen parametri on listan asetuksia, kuten reitin nimi, oletusmalli taustalle ja malli listan sisällölle.

```
this.Pages = new Meteor.Pagination(Stocks, {
  itemTemplate: "Stock",
  route: "/stocks/",
  homeRoute : "/stocks/",
  router: "iron-router",
  routerTemplate: "Stocks",
  routerLayout: "MasterLayout",
  sort: {
    name: 1
  },
  templateName: "Stocks"
});
```

KUVA 38 Meteor-pages:n lisäys

Malleja varten tarvittiin HTML-tiedosto ja sen vuoksi luottiin stocks.html niminen tiedosto, johon määritettiin kaksi mallia (kuva 39).

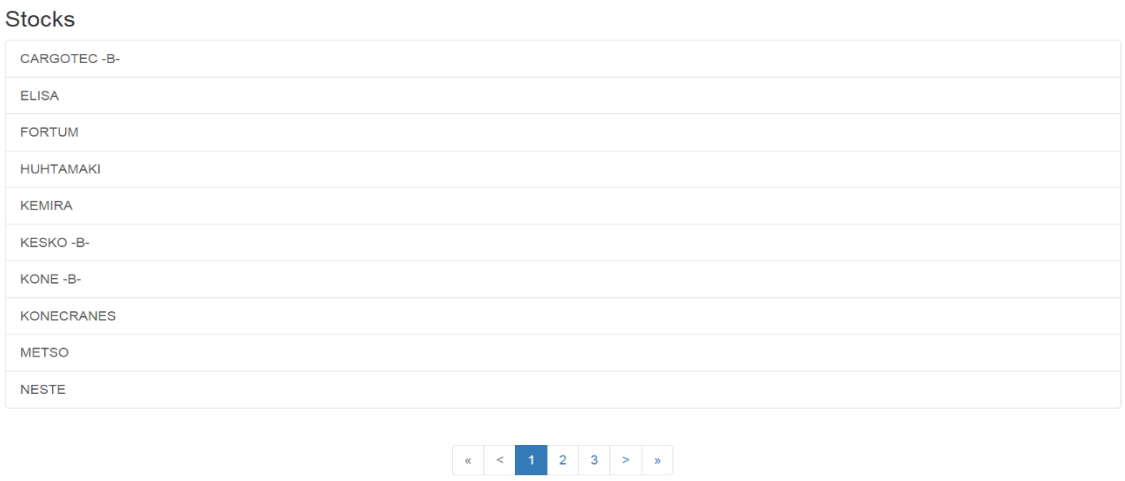
```
<template name="Stocks">
  <h3>Stocks</h3>
  <div class="list-group">
    {{> pages}}
  </div>
  {{> pagesNav}}
</template>

<template name="Stock">
  <a href="/stock/details/{{symbol}}" class="list-group-item">{{name}}</a>

</template>
```

KUVA 39. Stocks ja Stock mallit

Määritettiin kaksi mallia Stocks ja Stock. Stocks-malli sisältää periaatteessa meteor-pages ominaisuudet pages, johon tulee Stock-mallin mukaan jokaisen osakkeen nimi ja pagesNav, joka lisää listalle sivunumeroinnin. Jokaiselle osakkeelle on lisätty linkki ”/stock/details/{{symbol}}”. Kaksoissulkujen sisällä on lyhenne osakkeen nimestä, joka siirtyy seuraavalle sivulle osakkeen painamiseen jälkeen. Lisätty lista näyttää kuvan 40 mukaiselta.



KUVA 40. Lista osakkeista

6.11.2 Osakkeen tietoja ja viivakaavio

Ensin tuli määrittää routes.js-tiedostoon reitti jokaiselle osakkeelle. Reitti toimii dynaamisesti tässä tapauksessa sen mukaan, mitä syötetään linkkiin osakkeen nimeksi ”:stockName” (kuva 41). Syötetty arvo tarkistetaan tietokannassa. Näin varmistetaan, että kyseisen niminen osake on olemassa.

```
Router.route('stock/details/:stockName', {
  name: 'StockDetail',
  template: 'StockDetail',
  controller: 'StockDetailController',
  action: 'action',
  where: 'client'
});
```

KUVA 41. Reitti määritetty routes.js-tiedostossa

Reitittimille määritettiin nimi ja malli sanalla StockDetail ja sen lisäksi luottiin stockDetail.html- ja stockDetail.js-tiedostot.

Osakkeiden tietoja haettiin käyttämällä Meteorin yhtä pakettia, joka käyttää hyväksi yhden taloussivun rajapintaa. Kaikki kyselyt tehtiin palvelinpuolella tiedostossa methods.js, johon määritettiin kaksi funktiota ”getStock” ja ”getStockHistory”. Ensimmäisen funktion tarkoitus on hakea nimen mukaan osakkeen viimeinen päätöskurssi ja kurssin muutos. Toisen funktion tarkoitus on hakea yhden vuoden tietoa osakkeesta, jota käytetään hyväksi viivakaavion luonnissa.

Kutsumalla palvelinpuolen funktioita saadaan osakkeen tietoja, joita hyödynnetään stockDetail.html-tiedostossa (kuva 42). Viivakaavio luodaan käyttämällä Chart- ja Scatter-paketteja, joilla pystyy esittämään monenlaisia kaavioita eri tarkoituksia varten. Tiedostoon stockDetail.js määritettiin auttaja-funktio nimellä ”stockInformation”, joka palauttaa käyttöliittymälle tiedot osakkeesta eli viimeisen päätöskurssin ja kurssinmuutoksen.

```
<template name="StockDetail">
    <canvas id="historyChart" width="700" height="200"></canvas>
    {{#with stockInformation}}
    <h2>{{name}}</h2>
      <table class="table">
        <tbody>
          <tr>
            <td>Previous close</td>
            <td>{{previousClose}}</td>
          </tr>
          <tr>
            <td>Change</td>
            <td>{{change}}</td>
          </tr>
        </tbody>
      </table>
    {{/with}}
</template>
```

KUVA 42. StockDetail-template määritetty

Viivakaavion luontia varten haettiin HTML:stä viittaus canvas-elementtiin, jota käytettiin kaaviota luotaessa (kuva 43). Funktiolle annettiin kohde, mihin tulee kaavio ja data, josta luodaan kaavion sisältö eli päivämäärä vaakatasossa ja päätöskurssi pystytasossa. Kaaviolle pystyy antamaan lisäasetuksia, ja tässä tapauksessa määritettiin päivämääräasteikko vaakatasolle ja päivämäärän muoto.

```
  var ctx = document.getElementById("historyChart").getContext("2d");
  var myDateLineChart = new Chart(ctx).Scatter(data, {
    bezierCurve: true,
    showTooltips: true,
    scaleShowHorizontalLines: true,
    scaleShowLabels: true,
    scaleType: "date",
    scaleDateTimeFormat: "mmm d, yyyy"

  });
};
```

KUVA 43. Viivakaavion luonti

Toteutettu viivakaavio ja osakkeen tiedot näkyvät kuvassa 44.

KONE -B-

Previous close	41.44
Change	-0.06

KUVA 44. Viivakaavio ja osakkeen tiedot

7 POHDINTA

Työn tarkoitus oli tutustua Meteor-sovelluskehykseen ja kehittää sillä web-sovelluksen prototyyppi. Meteor-sovelluskehykseen tutustuttiin käymällä läpi opetusmateriaalia siihen liittyvää ja tutustumalla sen periaatteisiin ja ainutlaatuisiin ominaisuuksiin. Tutustumiseen jälkeen kehitettiin web-sovelluksen prototyyppi.

Prototyypin tavoite oli kehittää virtuaaliympäristö, jossa opiskelijat harjoittelisivat sijoittamista oikeilla käytännöillä ilman negatiivisia seuraamuksia. Prototyypin toinen tavoite oli myös tarjota opiskelijoille sijoittamisesta oppimismateriaalia, jota he voivat käyttää hyväksi silloin, kun he harjoittelevat web-sovelluksen ympäristössä.

Web-sovellukseen saatiin paljon perustoimintoja valmiiksi, kuten käyttäjätilijärjestelmä, käyttöliittymän tärkeitä osia ja navigaatio, käyttämällä pelkästään Meteoria varten kehitettyjä paketteja. Suurin osa toteutetuista ominaisuuksista oli perustoimintoja, joita käytetään melkein kaikissa nykypäivän web-sovelluksissa. Kehityksen loppupuolella saatiin lisättyä web-sovellukseen yksittäisiä ominaisuuksia, jotka vastasivat prototyypin tavoitetta.

Projektin aikana ei ehditty lisätä kaikkia ominaisuuksia web-sovellukseen, koska kehittäminen Meteorilla poikkesi paljon vanhoista web-tekniikoista ja sen vuoksi jokaisen ominaisuuden lisääminen edellytti lisää tutkimista. Lopputulokseen voidaan silti olla tyytyväisiä, koska saatiin kaikki ominaisuudet lisättyä oikealla tavalla käyttämällä suositeltavia menetelmiä.

Web-sovelluksen kehittäminen Meteorilla oli mielenkiintoista, koska se oli minun ensimmäinen web-sovelluksen toteutukseni uudella web-tekniikalla, josta sain ensivaikutelman uusista kehitysmenetelmistä ja web-sivun päivityksestä dynaamisella tavalla.

LÄHTEET

1. Beal, Vangie. Web - World Wide Web. Webopedia. Saatavissa: http://www.webopedia.com/TERM/W/World_Wide_Web.html. Hakupäivä 6.11.2015.

2. Khochare, Nilesh – Chalurkar, Satish – Meshram, B.B. 2013. Web Application Vulnerabilities Detection Techniques Survey. IJCSNS International Journal of Computer Science and Network Security, VOL.13 No.6, June 2013. Saatavissa: http://paper.ijcsns.org/07_book/201306/20130611.pdf. Hakupäivä 5.11.2015.

3. Nations, Daniel. 2013. Web Applications. About tech. Saatavissa: http://webtrends.about.com/od/webapplications/a/web_application.htm. Hakupäivä 5.11.2015.

4. Beal, Vangie. HTML - HyperText Markup Language. Webopedia. Saatavissa: http://www.webopedia.com/TERM/H/HTML.html. Hakupäivä 1.2.2016.

5. Lyons, Dan. 2013. What Is CSS? [FAQs]. HubSpot. Saatavissa: http://blog.hubspot.com/marketing/what-is-css-faq-ht. Hakupäivä 1.2.2016.

6. Introduction. 2015. Mozilla Developer Network. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction. Hakupäivä 1.2.2016

7. Aston, Ben. 2015. A brief history of JavaScript. Mozilla Developer Network. Saatavissa: https://medium.com/@benastontweet/lesson-1a-the-history-of-javascript-8c1ce3bffb17#.az30bznrt. Hakupäivä: 2.2.2016

8. Padolsey, James. 2009. JavaScript and the DOM Series: Lesson 1. Saatavissa: http://code.tutsplus.com/tutorials/javascript-and-the-dom-series-lesson-1--net-3134. Hakupäivä 3.2.2016.

9. Resig, John – Ferguson, Russ – Paxton, John. 2015. Pro JavaScript Techniques Second Edition. New York: Apress.

10. Hein, Richard. 2012. 6 reasons you should be using jQuery. Saatavissa: http://www.javaworld.com/article/2078613/java-web-development/6-reasons-you-should-be-using-jquery.html. Hakupäivä 5.2.2016.

11. Gimmer, Christopher. 2014. Top 5 Reasons to use Bootstrap. Saatavissa: https://bootstrapbay.com/blog/reasons-to-use-bootstrap/. Hakupäivä 5.2.2016.

12. Dayley, Brad. 2014. Node.js, MongoDB, and AngularJS Web Development. New Jersey: Addison-Wesley.

13. Tsonev, Krasimir. 2015. Node.js By Example. Birmingham: Packt Publishing Ltd.

14. Martinez, Daniel Pecos. 2013. NODE.JS AND V8 HISTORY. Saatavissa: http://nodegeek.net/2013/12/18/nodejs-v8-history/. Hakupäivä 6.11.2015.

15. Lehmann, Marc. libeio Saatavissa: http://software.schmorp.de/pkg/libeio.html. Hakupäivä 3.2.2016.

16. pcg1. libev. 2000. Saatavissa: http://freecode.com/projects/libev. Hakupäivä 3.2.2016.

17. npm (software). 2015. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Npm_(software). Hakupäivä 6.11.2015.

18. Hochhaus, Stephan – Schoebel, Manuel. 2015. Meteor In Action. New York: Manning Publications.

19. Meteor. Saatavissa: https://www.meteor.com. Hakupäivä 8.2.2016.

20. Rouse, Margaret. 2014. MongoDB. Saatavissa: http://searchdatamanagement.techtarget.com/definition/MongoDB. Hakupäivä 6.2.2016.